CS123 - Projects in Database Systems

Spring 2017
Course Overview

• Purpose:
  • Give students an opportunity to do some database system programming
  • (Not focused on applications that use databases)
  • Students can work individually, or in groups of any size

• Examples:
  • Add features to NanoDB
  • Create another database system from scratch, e.g. a graph DB, XML DB, a pure relational DB, etc.
  • Implement a feature on some open-source database

• [http://courses.cms.caltech.edu/cs123/](http://courses.cms.caltech.edu/cs123/)
Course Expectations (1)

- Choose an area of database systems that is interesting to you, and implement a project in it.

- Course is rated at 9 units, so all students should be working ~9 hours every week on this class.
  - A substantial part of this should be actual software development (design, implementation, testing, etc.).

- Must use version-control to manage your code.
  - e.g. GitHub, Bitbucket, etc.
  - Please be careful about sharing full NanoDB sources...
Course Expectations (2)

- Must write comprehensive docs for your work
  - Document the code you write, details on getting the project sources, and building/using the project

- Participate in CMS Meeting of the Minds
  - [http://cms.caltech.edu/research/meeting_of_the_minds](http://cms.caltech.edu/research/meeting_of_the_minds)
  - Friday, May 19th from 1:30pm-5:00pm
  - Make a poster, demo your work if you like, talk about your project with lots of interesting people!
  - Poster printing will be reimbursed by the CMS department

- Submit a final report describing your work
  - Mainly to point Donnie at the sources and how to try them out
Project Proposal

• Week 1: Write a project proposal (due Friday at 5pm)
  • A specific description of overall goals/features
  • Implementation details:
    • If extending an existing project, describe the project, where to get the code, etc.
    • Programming language you will use for the project
    • Version control repository you will use (e.g. git/svn/hg/bzr/...)
    • Where the repository will be hosted (e.g. CS cluster, github)
  • Outline/schedule of how you intend to complete the project in your limited 10-week timeframe
    • Break down implementation process into a sequence of tasks
    • No task longer than ~1 week of coding time
Project Proposal and Grades (1)

- I will review your project proposals for suitability
  - I will give you feedback if the project is acceptable, or if it is too much or too little work
  - Aside: if you need the full NanoDB sources, let me know

- Refer back to course expectations for guidance
  - ~9 hours a week of work per student, implementing something you didn’t already know about, etc.

- Your grade will be assigned based on:
  - How well your project matches up to this expectation
  - How hard you actually work on your project
Your overall grade will be lowered if:

- You don’t complete the features you specify
  - (Barring unforeseen implementation challenges, etc.)
- Your code isn’t clean and well commented
- You don’t use version-control, or you mismanage it
- You miss a milestone, or don’t do a good job on it
- Your final presentation isn’t good
- Your documentation isn’t complete
- etc.
Project Proposal and Grades (3)

- Your overall grade will be increased if:
  - You complete other features in addition to your project
  - You have substantial unit-testing in place
  - Your work will make a future iteration of CS122 easier to teach, or better, or both! 😊
  - etc.
Weekly Status Updates

- For non-milestone weeks: status updates

- These are required
  - Can either do these as a class, or can set up individual project meetings with Donnie

- Similar to scrum standups:
  - What you worked on the last week, and what you accomplished.
  - Unresolved issues you have run into.
  - What you plan to work on during the next week.
Milestone 1: Can you build it?

- **Week 3: Milestone 1**
  - Project should be well underway at this point
  - Goal: make sure your basic development framework/infrastructure is in place

- **Demo:**
  - How to check out code from your repository
  - How to build your project, generate docs, run unit-tests (if you have them at this point)

- **Submit these things to Donnie:**
  - Location of repository, and steps to follow to complete demo
  - Any additional libraries/dependencies I would need, outside of what is in the repository
Milestone 2: Development Checkpoint

- **Week 7: Milestone 2**
  - CMS Meeting of the Minds at the end of this week!

- Participate in the poster session:
  - Motivation for your project: What is it supposed to do? Why did you choose the project? What is cool about it?
  - Give an architectural overview of your project
  - Maybe: How project’s features map into the codebase
  - If you have cool stuff to demo, you can do that!

- To facilitate this, you should have a draft of your poster for your Week 6 meeting, for feedback from Donnie
Milestone 3: Project Complete (1)

- **Week 10/Finals Week:** Milestone 3
  - Your project should basically be complete at this point
  - Make sure the project is as bug-free and well documented as possible
  - If you had any measurements to take, they should be done

- **Presentation to Donnie:**
  - Run through a simple demo of all major components of your project
  - If you have any graphs or figures to present, do that too
  - Also, submit a brief final report to Donnie
Milestone 3: Project Complete (2)

- Also need to give Donnie a brief report on your project
  - e.g. as a PDF file

- Contents:
  - A description of your project
    - What you implemented, the major features, and any important architectural details
    - If a group project, description of each person’s specific focus
  - If you ran into significant challenges, what were they, and how did you overcome them?
  - How to get the code, how to build it (including building the docs), how to run the tests
  - A script for a demo for Donnie to try out your project
Ideas for NanoDB-Based Projects

- Things that will help with future iterations of CS122:
  - Plan-costing improvements
    - Rework the plan-costing to include disk-seek estimates, update column statistics to reflect predicates, etc.
    - Rework plan-costing to delegate to the storage file-format implementation for some format-specific estimates
    - Update stats-collection mechanism to be generic, and incorporate into B^+ tree implementation
Ideas for NanoDB-Based Projects (2)

- Things that will help with future iterations of CS122:
  - Multiple-client improvements
    - Get the client/server code working so that multiple clients can connect to a single NanoDB server
    - Write test classes that can use this code to spin up a NanoDB server in a separate JVM, and then connect to it
    - Can even write a tool to script interactions with NanoDB to exercise concurrency-control operations
    - There are still plenty of synchronization issues in NanoDB that can be ironed out, if you are interested in that kind of thing
Ideas for NanoDB-Based Projects (3)

- Things that will help with future iterations of CS122:
  - Constraint enforcement and index support
    - Currently, constraint enforcement depends on indexes being enabled; factor this out
  - Index-scan implementation
  - Nested-loop with index optimizations
  - Sort-merge join implementation
  - Selinger-style plan optimizer that takes advantage of indexes and ordered subplan-results
Ideas for NanoDB-Based Projects (4)

- Things that will help with future iterations of CS122:
  - A benchmark data-set and SQL queries
    - Queries that exercise various kinds of plan optimizations
    - Subqueries in IN clauses, EXISTS clauses, scalar subqueries in SELECT/WHERE expressions
    - Nested subqueries in FROM clauses where outer predicates can be pushed down
    - Grouping/aggregates that support predicate pushdown
    - Grouping/aggregates with “covering indexes” etc.
Ideas for NanoDB-Based Projects (4)

- Other fun NanoDB things:
- Complete the implementation of table-returning functions
- Implement SQL set-operations: UNION, INTERSECT, EXCEPT
- Implement Common Table Expressions (CTEs) for the WITH clause, temporary tables, etc.
- Implement lock-based concurrency control
- Implement multiversion timestamp-ordering concurrency control